Control Structures

For Loop

A for loop uses a variable (called a ***counter*** or ***index***) to keep track of how many times the loop has executed, and it stops when the counter reaches a certain number.

Syntax

```
for (initialization; test_condition; iteration_statement)
{
Statement(s) to be executed if test condition is true
}
```

To display the series 0,2,4….. 10th term

```
<html>
<body>
<script type="text/javascript">
var a=1,b=0;
for (a=1;a<=10;a++)
{
document.write(b);
b=b+2;
document.write("<br/>");
}
</script>
</body>
</html>
```

Output:

0
2
4
6
8
10
12
14
16
18

To display the series 1,2,3….. 10^{th} term

Output:

```
<html>
<body>
<script type="text/javascript">
var a=1,b=1;
for (a=1;a<=10;a++)
{
document.write(b);
b=b+1;
document.write("<br/>");
}
</script>
</body>
</html>
```

To display the series 1,4,9….. 6th term

Output:

```
<html>
<body>
<script type="text/javascript">
var a=1,b=1;
for (a=1;a<=10;a++)
{
document.write(b*b);
b=b+1;
document.write("<br/>");
}
</script>
</body>
</html>
```

To display the series

```
<html>
<body>
<script type="text/javascript">
var a=1,b=1;
for (a=1;a<=5;a++)
{
document.write(b);
b=b*10+1;
document.write("<br/>");
}
</script>
</body>
</html>
```

Output:

1
11
111
1111
11111

Control Structures

- JavaScript supports the following forms of **if..else** statement:

☞ if statement

☞ if...else statement

☞ if...else if... statement

If statement

- The '**if**' statement is the control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

```
if (expression)
{
Statement(s) to be executed if expression is true
}
```

- Write a program in java script to ask a number from user and test whether the number is negative.

```
<html>
<body>
<script type="text/javascript">
var a;
a=prompt("Enter the number");
if(a<0)
document.write("The number is negative");
</script>
</body>
</html>
```

If else statement

- If ….**else** statement is used to execute a block of code if the condition is true and another code if the condition is false.

Syntax

```
if (condition) {
    block of code to be executed if the condition is true
} else {
    block of code to be executed if the condition is false
}
```

Example

```
<html>
<body>
<script type="text/javascript">
var a;
a=prompt("Enter the number");
if(a>0)
{
document.write("The number is positive");
}
else
{
document.write("The number is negative");
}
</script>
</body>
</html>
```

If else if else statement

The '**if...else if...**' statement is an advanced form of **if...else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

```
if (expression 1){
Statement(s) to be executed if expression 1 is true
}else if (expression 2){
Statement(s) to be executed if expression 2 is true
}else if (expression 3){
Statement(s) to be executed if expression 3 is true
}else{
Statement(s) to be executed if no expression is true
}
```

❑ JavaScript program to check whether the number is positive, negative or neutral

```
<html>
<body>
<script type="text/javascript">
var a;
a=prompt("Enter the number");
if(a>0)
{
document.write("The number is positive");
}
else if(a<0)
{
document.write("The number is negative");
}
else
{
document.write(" The number is zero")
}
</script>
</body>
</html>
```

Switch Case

Switch case statement select one of many blocks of code to be executed.

Syntax

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        code block
}
```

Break Keyword

When JavaScript reaches a **break** keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

Default Keyword

The **default** keyword specifies the code to run if there is no case match

JavaScript program to display day of a week

```
<html>
<body>
<script type="text/javascript">
var day;
day=prompt("Enter the day");
switch (eval(day))
{
case 1:
document.write("Sunday");
break;
case 2:
document.write("Monday");
break;
case 3:
document.write("Tuesday");
break;
case 4:
document.write("Wednesday");
break;
case 5:
document.write("Thursday");
break;
case 6:
document.write("Friday");
break;
case 7:
document.write("Saturday");
break;
default:
document.write("Invalid choice");
}
</script>
</body>
</html>
```

While Statement

While loop execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false**, the loop terminates.

Syntax:

```
while (condition) {
    code block to be executed
}
```

To display the series 1,2,3,……10^{th} term

```
<html>
<body>
<script type="text/javascript">
var a=1;
while(a<=10)
{
document.write(a);
a=a+1;
document.write("<br/>");
}
</script>
</body>
</html>
```

Do...while loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do
{
  code block to be executed
}
while (condition);
```

example

```
<html>
<body>
<script type="text/javascript">
var a=1;
do
{
document.write(a);
a=a+1;
document.write("<br/>");
}
while(a<=10);
</script>
</body>
</html>
```

Functions

- A function is simply a block of code with a name, which allows the block of code to be called by other components in the script to perform certain tasks.
- Functions can also accept parameters that they use to complete their task.

Types of Functions

Built in functions

User define functions

Built in functions

- The functions present in the program itself is called built in functions. It makes the task easier to perform.
- Some of them are eval(), parseInt() and parseFloat()

Eval(): it is used to convert a string expression to a numeric value.

eg. Var sum=eval(“4*5+5”)

it will return the result as 25 in variable sum. If number is not contained the argument of eval(), it gives error message.

- **ParseInt():** it is used to return the first integer part of string.

Eg: parseint(“123 Adarsha!”)

It will return the result as 123 only the first integer part of the string. If it does not have an integer as the first part of string, it gives error NAN (not a number)

ParseFloat (): it is used to return the first floating point number part of the string.

Eg: ParseFloat(“123.45 Hello!”)

it will return the result as 123.45. It returns NAN, if a first part of the string is not the floating point number.

User define functions

- In addition to using the functions provided by java script, you can also create and use your own functions.
- Syntax for creating functions in javascript

```
Function name-of-function(arg1,arg2,…..arg)
{
Block of code
}
```

Calling functions

- There are two common ways to call a function.
  - From another functions
  - From an event handler
- Calling a function is simple. You have to specify its name followed by the pair of parenthesis.

  <Script type="text/javascript">

  Function name-of-function(arg1,arg2…..)

  </script>

```
<html>
<head>
<script type="text/javascript">
function  welcomeMessage()
{
document.write("welcome to MMAMC BIT");
}
</script>
</head>
<body>
<h1> MMAMC COLLEGE</h1>
<h3>Testing the function </h3>
<script type="text/javascript">
welcomeMessage();
</script>
</body>
</html>
```

Fibonacci series using function

```
<html>
 <head>
        <script type="text/javascript">
        function fibnoic()
        {
                i=0;
                j=1;
                for(n=0;n<=12;n++)
                {
                document.write(i + ", ");
                        k=i+j;
                i=j;
                j=k;
                        }
                }

                </script>
</head>
                <body>
                <h3> Fibonic series</h3>
                <script type="text/javascript">
                fibnoic();
                        </script>
                </body>
                </html>
```

Alert box

- An alert box is often used if you want to make sure information comes through the user. When an alert box pops up the user will have to click Ok to proceed.

Syntax:

Alert (“some text”)

Example

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("i am an alert box!");
}
</script>
</head>
<body>
<input type="button" onclick="show_alert()" value="show alert box"/>
</body>
</html>
```

Confirmation Box

- A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either **Ok** or **Cancel** to proceed. If the user clicks **Ok**, the box returns true. If the user click **Cancel**, the box returns false.

Syntax:

confirm(“sometext”)

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if(r==true)
{
document.write("You pressed Ok!")
}
else
{
document.write("You pressed Cancel!")
}
}
</script>
</head>
<body>
<input type="button" onclick="show_confirm()"
value="show confirm box"/>
</body>
</html>
```

Prompt box

- A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either **Ok** or **Cancel** to proceed after entering an input value. If the user clicks **Ok** the box returns the input value. If the user clicks **Cancel** the box returns null.

syntax :

Prompt(“Some text”,”default value”)

Example

```
<html>
<head>
<script type="text/javascript">
var name=prompt("Please enter your name",“Pop");
</script>
</head>
<body>
<script type="text/javascript">
document.write("Hello "+name  + " Good morning!");
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Prompt</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  let text;
  let person = prompt("Please enter your name:", "peter");
  if (person == null || person == "") {
    text = "User cancelled the prompt.";
  } else {
    text = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

Javascript Objects

- JavaScript is an Object Oriented Programming(OOP) language. An OOP language allows you to define your own object and make your own variable types. An object is just a special kind of data.
- An object has **Properties** and **Methods**

- **Properties:** Properties are the values associated with an object.

```
<html>
<body>
<script type="text/javascript">
var text=“MMAMC"
document.write(text.length);
</script>
</body>
</html>
```

We are using the length property of the string object to return the number of character in a string.

• **Methods:** Methods are the actions that can be performed on objects.

```
<html>
<body>
<script type="text/javascript">
var str=“Biratnagar"
document.write(str.toUpperCase())
</script>
</body>
</html>
```

We are using the touppercase () method of the string object to display a text in uppercase letter.

Array object in JavaScript

- An array is a special variable, which can hold more than one value, at a time. An array can hold all your variables value under a single name and you can access the values by referring to the array name. Each element in the array has its own id so that it can be easily accessed.
- **Declaring an Array**
- **Syntax: var Arrayname=new array()**

  var arrayname=[list of items]

Example

```
<html>
<body>
<script type="text/javascript">
var p;
var fren=["Biraj","Anil","Anita","Nita"]
for (p=0;p<=3;p++)
document.write(fren[p],"<br>")
</script>
</body>
</html>
```

Example

```
<html>
<head>
<title></title>
<script langauge="javascript">
function my()
{
a=new Array()
for(i=0;i<10;i++)
{
a[i]=prompt("enter the 10 names","");
}
document.write(a.join());
}
</script>
</head>
<body onload="my()">
</body>
 </html>
```

```
<html>
<head>
<title>
</title>
</head>
<body>
<script language='javascript'>
i=0;
function changebg()
{
a=new Array();
a[0]="RED";
a[1]="BLACK";
a[2]="GREEN";
a[3]="WHITE";
a[4]="BROWN";
a[5]="GREY";
a[6]="PURPLE";
a[7]="PINK";
a[8]="BLUE";
a[9]="YELLOW";
a[10]="VIOLET"
document.bgColor=a[i];
i++;
}
setInterval("changebg()",2000)
</script>
</body>
</html>
```

Java script sorting arrays

- There are two methods for arranging elements of an array.
- Sort(): it sorts the element of an array in ascending order.
- Reverse():It reverse the element of an array.

Example To display in ascending order

```
<html>
<body>
<script type="text/javascript">
var fren=["Biraj","Anil","Anita","Nita","kamal"]
document.write(fren.sort(),"<br>")
</script>
</body>
</html>
```

Example To display in reverse.

```
<html>
<body>
<script type="text/javascript">
var fren=["Biraj","Anil","Anita","Nita","kamal"]
document.write(fren.reverse())
</script>
</body>
</html>
```

User define object in Javascript

- An object is a special kind of data, with a collection of properties and methods.
- Example: A person is an object. Properties are the valued associated with the object. The persons' properties included name, weight, age, skin tone, eye color etc. All person have these properties, but the values of those properties will differ from person to person.
- Objects also have methods. Methods are the actions that can be performed on objects. The persons' methods could be eat(), sleep(), work(), play() etc.

Accessing property of an object

Syntax: Objectname.propname

```
<html>
<body>
<script type="text/javascript">
var person={firstname:"Biraj", lastname:"Thapa",age:20,eyecolor:"black"}
document.write(person.firstname+" is "+person.age+ "years old")
document.write("<br>", "His eye color is " +person.eyecolor)
</script>
</body>
</html>
```

The **this** Keyword

- In JavaScript, the thing called **this**, is the object that "owns" the JavaScript code.
- The value of **this**, when used in a function, is the object that "owns" the function.

Accessing object Method

Syntax: Objectname.methodname()

```
<html>
<body>
<script type="text/javascript">
var person={
firstname:"Biraj",
lastname:"Thapa",
age:20,
eyecolor:"black",
fullname:function(){
return this.firstname+ " " +this.lastname
}
};
document.write(person.fullname());
</script>
</body>
</html>
```

Form Validation

- Forms are usually used for accepting data from the user like name, username, password, numbers etc.
- Form validation is the process of checking that a form has been filled in correctly before it is processed.
- For example if your form has a box for the user to type the email address, you might want your form handler to check that they have filled in the address before you deal with the rest of the form.
- There are two methods for validating forms:
- **Server side (using CGI scripts, ASP etc)**
- **Client side (usually done using javascript)**

- **Server side** : Server side validation is more secure but more tricky to code and it also increases load of server computer.
- **Client side**: Client side validation is easier to do and quicker too and it also decreases the load of server computer.

Example

```
<html>
<head>
<script>
function validateForm() {
var x =
document.forms["myForm"]["fname"].value;
    if (x == "")
            {
        alert("Name must be filled out..");
        return false;
    }
}
</script>
</head>
<body>
<form name="myForm" onsubmit="return
validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Write Java script code to validate username and password

```
<html>
<head>
<script>
function validate() {
      if (document.myform.username.value=="")
{
      alert("Please provide your name");
document.myform.username.focus();
      return false;
   }
if (document.myform.password.value=="")
{
 alert("Please provide your password");
document.myform.password.focus();
      return false;
}
return(true);
}
</script>
</head>
<body>
<form action="/cgi-bin/test.cgi"
name="myform" onsubmit="return
(validate());">
```

```
<table border=1>
<tr>
<td align=right>username</td>
<td><input type=text name=username></td>
</tr>
<tr>
<td align=right>password</td>
<td><input type=password name=password></td>
</tr>
<tr>
<td colspan=2 align="center">
<input type=submit value=submit>
</td>
</tr>
</table>
</form>
</body>
</html>
```

Note:

```
const username = document.getElementById('username').value;
const password = document.getElementById('password').value;
```

Extract the values typed into the username and password fields by the user and store them in the username and password variables for further validation or processing.

❑ Username and Password validation:

Condition:

Username: Must be at least 5 characters long.

Password: Must be at least 8 characters long.

Code:

```
<!DOCTYPE html>
<head>
   <title>form Validation</title>
   <script>
      function validateForm() {
         // Get the input values
 const username = document.getElementById('username').value;
 const password = document.getElementById('password').value;
 if (username.length < 5)
        {
 alert("Username must be at least 5 characters long.");
            return false;
         }
```

Code:

```
if (password.length < 8)
                        {
        alert("Password must be at least 8 characters long.");
            return false;
          }
          alert("Form is valid!");
          return true;
       }
   </script>
</head>
<body>
   <h2>Login Form</h2>
   <form onsubmit="return validateForm()">
    <label for="username">Username:</label>
   <input type="text" id="username" name="username" required><br><br>
 <label for="password">Password:</label>
<input type="password" id="password" name="password" required><br><br>
<input type="submit" value="Submit">
   </form>
</body>
</html>
```

Username and Password validation:

Condition:

Username: Allows 5-20 characters, letters, numbers, and underscores

Password: Requires at least one uppercase letter, one lowercase letter, one digit, one special character, and must be at least 8 characters long.

Code:

```
<!DOCTYPE html>
<head>
<title>Form Validation</title>
    <script>
        function validateForm() {
 const username = document.getElementById('username').value;
 const password = document.getElementById('password').value;
 // Define validation criteria
 const usernameRegex = /^[a-zA-Z0-9_]{5,20}$/; // Only alphanumeric characters and
underscore, 5-20 characters
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$/; // At least 8 characters, one uppercase, one lowercase, one number,
one special character
```

Code:

```
// Validate username
        if (!usernameRegex.test(username)) {
   alert("Username must be 5-20 characters long, and contain only letters, numbers, or underscores.");
            return false;
        }
        // Validate password
        if (!passwordRegex.test(password)) {
     alert("Password must be at least 8 characters long, contain at least one uppercase letter, one lowercase letter, one number, and one special character.");
            return false;
        }
        // If both validations pass
        alert("Form is valid!");
        return true;
      }
   </script>
</head>
```

Code:

```
<body>
    <h2>Login Form</h2>
    <form onsubmit="return validateForm()">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Event

- Events are actions that can be detected by javascript. Every element on a web page has certain events which can tigger java script functions.

Attribute	Value	Description
Onload	*Script*	Script to be run when a document loads
Onunload	*Script*	Script to be run when a document unloads

Form Element Events **(Only valid in form elements)**

Attribute	Value	Description
Onchange	*Script*	Script to be run when the element changes
Onsubmit	*Script*	Script to be run when the form is submitted
Onreset	*Script*	Script to be run when the form is reset
Onselect	*script*	Script to be run when the element is selected
Onblur	*script*	Script to be run when the element loses focus
Onfocus	*script*	Script to be run when the element gets focus

Example on click event

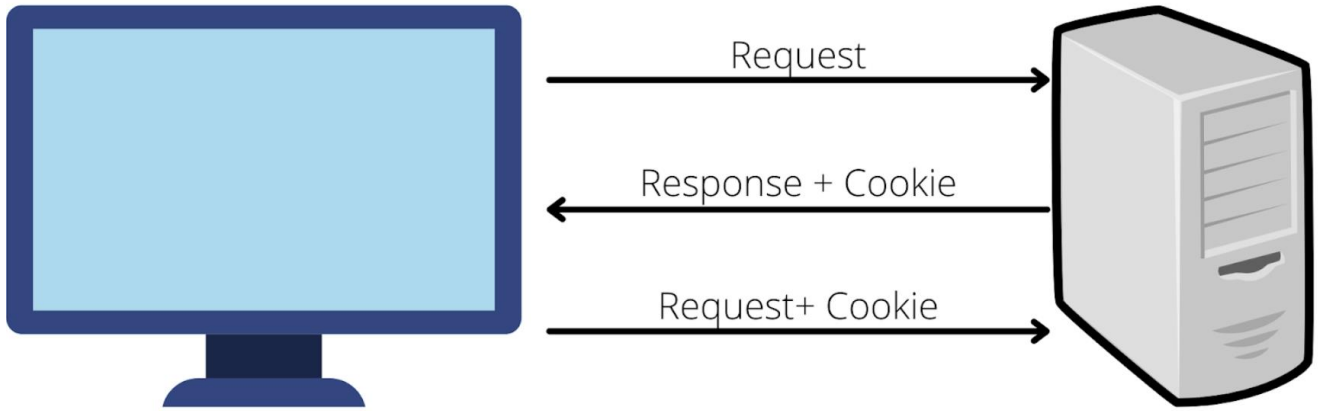
```
<html>
<head>
<script type="text/javascript">
function sayHello() {
document.write ("Hello World")
}
</script>
</head>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Handling Cookies

- Cookies are text files stored on the client computer that contains user information. Or **Cookies are piece of data stored in the user's web browser.**
- A cookie is often used to identify a user. The cookies are stored in the key-value pair inside the browser.
- Cookies may contain:
  - ✓ Username and Password
  - ✓ Links that you've clicked on
  - ✓ Your cart information for buying online
  - ✓ Visitor tracking information
  - ✓ Your computers general location in the world
  - ✓ Video's you've watched

Handling Cookies

✓How cookies works?



Web browser **Server**

- When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.
- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.

Handling Cookies

▪ How to create a Cookie in JavaScript?

- ✓ In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.
- ✓ The following syntax is used to create a cookie:

```
document.cookie="name = value";
```

Example:

```
document.cookie = "username=John";
```

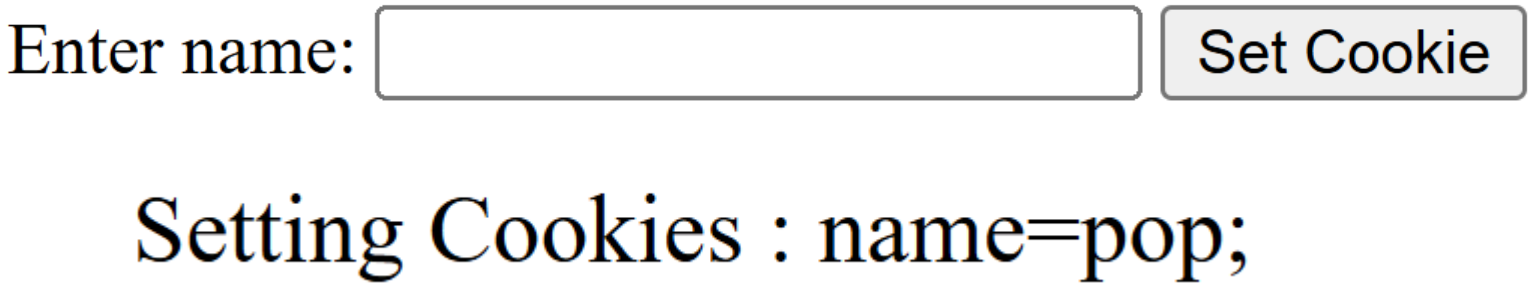
Note: By default, the cookie is deleted when the browser is closed:

- **Example:**

```
<html>
  <head>
    <script>
      function WriteCookie() {
        if( document.myform.customer.value == "" ) {
          alert("Enter some value!");
          return;
        }
   cookievalue = escape(document.myform.customer.value) + ";";
   document.cookie = "name=" + cookievalue;
   document.write ("Setting Cookies : " + "name=" + cookievalue );
        }
    </script>
  </head>
```

- **Example:**

```
<body>
    <form name = "myform" action = "">
      Enter name: <input type = "text" name = "customer"/>
      <input type = "button" value = "Set Cookie" onclick = "WriteCookie();"/>
    </form>
  </body>
</html>
```



- **Example:**

Note − Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript **escape()** function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding **unescape()** function when you read the cookie value.

Cookies vs. Sessions

Cookies	Sessions
1. Cookies are client-side text files that contains user information.	1. Sessions are Server-side files that contains user information.
2. Cookies are stored in the user's browser.	2. Sessions are not stored in user's browser.
3. A cookie can keep information in the user's browser until deleted	3. Session is that when you close your browser you also lose the information.
4. Cookie store limited amount of data i.e. 4Kb(4096 bytes)	4. It stores unlimited amount of data
5. Cookies can only store string	5. Store user's object in session
6. Less secure	6. More secure

▪ Error handling

- ✓ Error handling is very important in a flexible and widely used language like JavaScript. Errors directly affect the user experience because they can create a negative perception about the reliability and quality of the application. Therefore, correct error management increases application reliability and ensures that users have a smooth experience.
- ✓ Error handling in JavaScript is essential for building robust and user-friendly applications.

Error handling

- **Try-Catch Blocks: Error Handling**

  - ✓ **try-catch** blocks are the most basic structure used for error handling in JavaScript. Codes that are likely to cause errors are run in the **try** block, and these errors are caught in the **catch** block.

Error handling

- **Try-Catch Blocks: Error Handling**

```
<p id="demo"></p>
<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

Error handling

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Error Handling</h2>
<p>How to use <b>catch</b> to display an error.</p>
<p id="demo"></p>
<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
</body>
</html>
```

- ✓ In addition to **try and catch**, there is one more block in the error handling mechanism in JavaScript, namely **finally**. The finally block will still execute regardless of the outcome of the try-catch block.

```
Syntax:

try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of
the try / catch result
}
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript try catch</h2>
<p>Please input a number between 5 and 15:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p02"></p>
<script>
function myFunction() {
  const message = document.getElementById("p02");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x.trim() == "")  throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 15)   throw "is too high";
    if(x < 5)  throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
</script>
</body>
</html>
```

❑ Basics of AJAX and jQuery

- AJAX stands for Asynchronous JavaScript and XML.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- Examples of applications using AJAX: Google Maps, Gmail, YouTube, and Facebook.
- AJAX is widely used in modern web applications, from chat apps to live search suggestions, and is integral to frameworks like **jQuery**, **React**, and **Angular**.

Introduction to JQUERY

- jQuery is a lightweight, "write less, do more", JavaScript library.
- The purpose of jQuery is to make it much easier to use JavaScript on your website.
  - The jQuery library contains the following features:
    - ✓ HTML/DOM manipulation
    - ✓ CSS manipulation
    - ✓ HTML event methods
    - ✓ Effects and animations
    - ✓ AJAX
    - ✓ Utilities

Introduction to JQUERY

- jQuery load() Method:

  The jQuery load() method is a simple, but powerful AJAX method.

  The load() method loads data from a server and puts the returned data into the selected element.

  Syntax:

  ```
  $(selector).load(URL,data,callback);
  ```